A Distributed Frank-Wolfe Framework for Learning Low-Rank Matrices with the Trace Norm

Wenjie Zheng, Aurélien Bellet, Patrick Gallinari

Noname manuscript No.
(will be inserted by the editor)

A Distributed Frank-Wolfe Framework for Learning Low-Rank Matrices with the Trace Norm

Wenjie Zheng · Aurélien Bellet · Patrick Gallinari

Received: date / Accepted: date

Abstract We consider the problem of learning a high-dimensional but low-rank matrix from a large-scale dataset distributed over several machines, where low-rankness is enforced by a convex trace norm constraint. We propose DFW-Trace, a distributed Frank-Wolfe algorithm which leverages the low-rank structure of its updates to achieve efficiency in time, memory and communication usage. The step at the heart of DFW-Trace is solved approximately using a distributed version of the power method. We provide a theoretical analysis of the convergence of DFW-Trace, showing that we can ensure sublinear convergence in expectation to an optimal solution with few power iterations per epoch. We implement DFW-Trace in the Apache Spark distributed programming framework and validate the usefulness of our approach on synthetic and real data, including the ImageNet dataset with high-dimensional features extracted from a deep neural network.

Keywords Frank-Wolfe algorithm · Low-rank learning · Trace norm · Distributed optimization · Multi-task learning · Multinomial logistic regression

1 Introduction

Learning low-rank matrices is a problem of great importance in machine learning, statistics and computer vision. Since rank minimization is known to be NP-hard, a principled approach consists in solving a convex relaxation of the problem where the rank is replaced by the trace norm (also known as the nuclear norm) of the matrix. This strategy is supported by a range of theoretical results showing that when the ground truth matrix is truly low-rank, one can recover it exactly (or accurately) from limited samples and under mild conditions (see Bach, 2008; Candès and Recht, 2009; Candès and Tao, 2010; Recht, 2011; Gross et al, 2010; Gross,

W. Zheng, P. Gallinari
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6
E-mail: first.last@lip6.fr

A. Bellet
INRIA
E-mail: first.last@inria.fr

2011; Koltchinskii et al, 2011; Bhojanapalli et al, 2016). Trace norm minimization has led to many successful applications, among which collaborative filtering and recommender systems (Koren et al, 2009), multi-task learning (Argyriou et al, 2008; Pong et al, 2010), multi-class and multi-label classification (Goldberg et al, 2010; Cabral et al, 2011; Harchaoui et al, 2012), robust PCA (Cabral et al, 2013), phase retrieval (Candes et al, 2015) and video denoising (Ji et al, 2010).

We consider the following generic formulation of the problem:

$$\min_{W \in \mathbb{R}^{d \times m}} F(W) = \sum_{i=1}^{n} f_i(W) \quad \text{s.t. } \|W\|_* \leq \mu, \tag{1}$$

where the f_i's are differentiable with Lipschitz-continuous gradient, $\|W\|_* = \sum_k \sigma_k(W)$ is the trace norm of W (the sum of its singular values), and $\mu > 0$ is a regularization parameter (typically tuned by cross-validation). In a machine learning context, an important class of problems considers $f_i(W)$ to be a loss value which is small (resp. large) when W fits well (resp. poorly) the i-th data point (see Section 2.3 for concrete examples).[1] In this work, we focus on the large-scale scenario where the quantities involved in (1) are large: typically, the matrix dimensions d and m are both in the thousands or above, and the number of functions (data points) n is in the millions or more.

Various approaches have been proposed to solve the trace norm minimization problem (1).[2] One can rely on reformulations as semi-definite programs and use out-of-the-shelf solvers such as SDPT3 (Toh et al, 1999) or SeDuMi (Sturm, 1999), but this does not scale beyond small-size problems. To overcome this limitation, first-order methods like Singular Value Thresholding (Cai et al, 2010), Fixed Point Continuation algorithms (Ma et al, 2011) and more generally projected/proximal gradient algorithms (Parikh and Boyd, 2013) have been proposed. These approaches have two important drawbacks preventing their use when the matrix dimensions d and m are both very large: they require to compute a costly (approximate) SVD at each iteration, and their memory complexity is $O(dm)$. In this context, Frank-Wolfe (also known as conditional gradient) algorithms (Frank and Wolfe, 1956) provide a significant reduction in computational and memory complexity: they only need to compute the leading eigenvector at each iteration, and they maintain compact low-rank iterates throughout the optimization (Hazan, 2008; Jaggi et al, 2010; Jaggi, 2013; Harchaoui et al, 2015). However, as all first-order algorithms, Frank-Wolfe requires to compute the gradient of the objective function at each iteration, which requires a full pass over the dataset and becomes a bottleneck when n is large.

The goal of this paper is to propose a *distributed version of the Frank-Wolfe algorithm* in order to alleviate the cost of gradient computation when solving problem (1). We focus on the Bulk Synchronous Parallel (BSP) model with a master node connected to a set of slaves (workers), each of the workers having access to a subset of the f_i's (typically corresponding to a subset of training points). Our contributions are three-fold. First, we propose DFW-Trace, a Frank-Wolfe algorithm relying on a distributed power method to approximately compute the

[1] More general cases can be addressed, such as pairwise loss functions $f_{i,j}$ corresponding to pairs of data points.

[2] Some methods consider an equivalent formulation where the trace norm appears as a penalization term in the objective function rather than as a constraint.

Algorithm 1 Centralized Frank-Wolfe algorithm to solve (2)

Input: Initial point $W^0 \in \mathcal{D}$, number of iterations T
for $t = 0, \ldots, T-1$ **do**
  $S^* \leftarrow \arg\min_{S \in \mathcal{D}} \langle S, \nabla F(W^t) \rangle$ ▷ solve linear subproblem
  $\gamma^t \leftarrow \frac{2}{t+2}$ (or determined by line search) ▷ step size
  $W^{t+1} \leftarrow (1-\gamma^t)W^t + \gamma^t S^*$ ▷ update
end for
Output: W^T

leading eigenvector with communication cost of $O(d+m)$ per pass over the dataset (*epoch*). This dramatically improves upon the $O(dm)$ cost incurred by a naive distributed approach. Second, we prove the sublinear convergence of DFW-Trace to an optimal solution in expectation, quantifying the number of power iterations needed at each epoch. This result guarantees that DFW-Trace can find low-rank matrices with small approximation error using few power iterations per epoch. Lastly, we provide a modular implementation of our approach in the Apache Spark programming framework (Zaharia et al, 2010) which can be readily deployed on commodity and commercial clusters. We evaluate the practical performance of DFW-Trace by applying it to multi-task regression and multi-class classification tasks on synthetic and real-world datasets, including the ImageNet database (Deng et al, 2009) with high-dimensional features generated by a deep neural network. The results confirm that DFW-Trace has fast convergence and outperforms competing methods. While distributed FW algorithms have been proposed for other classes of problems (Bellet et al, 2015; Moharrer and Ioannidis, 2017; Wang et al, 2016), to the best of our knowledge our work is the first to propose, analyze and experiment with a distributed Frank-Wolfe algorithm designed specifically for trace norm minimization.

The rest of this paper is organized as follows. Section 2 introduces some background on the (centralized) Frank-Wolfe algorithm and its specialization to trace norm minimization, and reviews some applications. After presenting some baseline approaches for the distributed setting, Section 3 describes our algorithm DFW-Trace and its convergence analysis, as well as some implementation details. Section 4 discusses some related work, and Section 5 presents the experimental results.

2 Background

We review the centralized Frank-Wolfe algorithm in Section 2.1 and its specialization to trace norm minimization in Section 2.2. We then present some applications to multi-task learning and multi-class classification in Section 2.3.

2.1 Frank-Wolfe Algorithm

The original Frank-Wolfe (FW) algorithm dates back from the 1950s and was originally designed for quadratic programming (Frank and Wolfe, 1956). The scope of the algorithm was then extended to sparse greedy approximation (Clarkson, 2010) and semi-definite programming (Hazan, 2008). Recently, Jaggi (2013) generalized

the algorithm further to tackle the following generic problem:

$$\min_{W \in \mathcal{D}} F(W), \tag{2}$$

where F is convex and continuously differentiable, and the feasible domain $\mathcal{D}$ is a compact convex subset of some Hilbert space with inner product $\langle \cdot, \cdot \rangle$.

Algorithm 1 shows the generic formulation of the FW algorithm applied to (2). At each iteration t, the algorithm finds the feasible point $S^* \in \mathcal{D}$ which minimizes the linearization of F at the current iterate W^t. The next iterate W^{t+1} is then obtained by a convex combination of W^t and S^*, with a relative weight given by the step size γ^t. By convexity of $\mathcal{D}$, this ensures that W^{t+1} is feasible. The algorithm converges in $O(1/t)$, as shown by the following result from Jaggi (2013).

Theorem 1 (Jaggi, 2013) *Let C_F be the curvature constant of F.*[3] *For each $t \geq 1$, the iterate $W^t \in \mathcal{D}$ generated by Algorithm 1 satisfies $F(W^t) - F(W^*) \leq \frac{2C_F}{t+2}$, where $W^* \in \mathcal{D}$ is an optimal solution to* (2).

Remark 1 There exist several variants of the FW algorithm, for which faster rates can sometimes be derived under additional assumptions. We refer to Jaggi (2013), and Lacoste-Julien and Jaggi (2015) for details.

From the algorithmic point of view, the main step in Algorithm 1 is to solve the linear subproblem over the domain $\mathcal{D}$. By the linearity of the subproblem, a solution always lies at an extremal point of $\mathcal{D}$, hence FW can be seen as a greedy algorithm whose iterates are convex combinations of extremal points (adding a new one at each iteration). When these extremal points have some specific structure (e.g., sparsity, low-rankness), the iterates inherit this structure and the linear subproblem can sometimes be solved very efficiently. This is the case for the trace norm constraint, our focus in this paper.

2.2 Specialization to Trace Norm Minimization

The FW algorithm applied to the trace norm minimization problem (1) must solve the following subproblem:

$$S^* \in \arg\min_{\|S\|_* \leq \mu} \langle S, \nabla F(W^t) \rangle, \tag{3}$$

where $W^t \in \mathbb{R}^{d \times m}$ is the iterate at time t and $S \in \mathbb{R}^{d \times m}$. The trace norm ball is the convex hull of the rank-1 matrices, so there must exist a rank-1 solution to (3). This solution can be shown to be equal to $-\mu u_1 v_1^\top$, where u_1 and v_1 are the unit left and right top singular vectors of the gradient matrix $\nabla F(W^t)$ (Jaggi, 2013). Finding the top singular vectors of a matrix is much more efficient than computing the full SVD. This gives FW a significant computational advantage over projected and proximal gradient descent approaches when the matrix dimensions are large. Furthermore, assuming that W^0 is initialized to the zero matrix, W^t can be stored in a compact form as a convex combination of t rank-1 matrices, which requires

[3] This constant is bounded above by $L \operatorname{diam}(\mathcal{D})^2$, where L is the Lipschitz constant of the gradient of F (see Jaggi, 2013).

$O(t(d+m))$ memory instead of $O(dm)$ to store a full rank matrix. As implied by Theorem 1, FW is thus guaranteed to find a rank-t whose approximation error is $O(1/t)$ for any $t \geq 1$. In practice, when the ground truth matrix is indeed low-rank, FW can typically recover a very accurate solution after $t \ll \min(d, m)$ steps.

We note that in the special case where the matrix W is square ($d = m$) and constrained to be symmetric, the gradient $\nabla F(W^t)$ can always be written as a symmetric matrix, and the solution to the linear subproblem has a simpler representation based on the leading eigenvector of the gradient, see Jaggi (2013).

2.3 Applications

We describe here two tasks where trace norm minimization has been successfully applied, which we will use to evaluate our method in Section 5.

Multi-task least square regression. This is an instance of multi-task learning (Caruana, 1997), where one aims to jointly learn m related tasks. Formally, let $X \in \mathbb{R}^{n \times d}$ be the feature matrix (n training points in d-dimensional space) and $Y \in \mathbb{R}^{n \times m}$ be the response matrix (each column corresponding to a task). The objective function aims to minimize the residuals of all tasks simultaneously:

$$F(W) = \frac{1}{2} \|XW - Y\|_F^2 = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i^T w_j - y_{ij})^2, \tag{4}$$

where $\|\cdot\|_F$ is the Frobenius norm. Using a trace norm constraint on W allows to couple the tasks together by making the task predictors share a common subspace, which is a standard approach to multi-task learning (see e.g., Argyriou et al, 2008; Pong et al, 2010).

Multinomial logistic regression. Consider a classification problem with m classes. Let $X \in \mathbb{R}^{n \times d}$ be the feature matrix and $y \in \{1, \ldots, m\}^n$ the label vector. Multinomial logistic regression minimizes the negative log-likelihood function:

$$F(W) = \sum_i \log\Big(1 + \sum_{l \neq y_i} \exp(w_l^T x_i - w_{y_i}^T x_i)\Big) = \sum_i \Big(-w_{y_i}^T x_i + \log \sum_l \exp(w_l^T x_i)\Big). \tag{5}$$

The motivation for using the trace norm is that multi-class problems with a large number of categories usually exhibit low-rank embeddings of the classes (see Amit et al, 2007; Harchaoui et al, 2012).

3 Distributed Frank-Wolfe for Trace Norm Minimization

We now consider a distributed master/slave architecture with N slaves (workers). The master node is connected to all workers and acts mainly as an aggregator, while most of the computation is done on the workers. The individual functions $f_1, \ldots, f_n$ in the objective (1) are partitioned across workers, so that all workers can collectively compute all functions but each worker can only compute its

own subset. Recall that in a typical machine learning scenario, each function f_i corresponds to the loss function computed on the i-th data point (as in the examples of Section 2.3). We will thus often refer to these functions as data points. Formally, let $I_j \subseteq \{1, \ldots, n\}$ be the set of indices assigned to worker j, where $I_1 \cup \cdots \cup I_N = \{1, \ldots, n\}$ and $I_1 \cap \cdots \cap I_N = \emptyset$. We denote by $F_j = \sum_{i \in I_j} f_i$ the local function (dataset) associated with each worker j, and by $n_j = |I_j|$ the size of this local dataset.

We follow the Bulk Synchronous Parallel (BSP) computational model: each iteration (*epoch*) alternates between parallel computation at the workers and communication with the master (the latter serves as a synchronization barrier).

3.1 Baseline Strategies

Before presenting our algorithm, we first introduce two baseline distributed FW strategies (each with their own merits and drawbacks).

Naive DFW. One can immediately see a naive way of running the centralized Frank-Wolfe algorithm (Algorithm 1) in the distributed setting. Starting from a common initial point W^0, each worker j computes at each iteration t its local gradient $\nabla F_j(W^t)$ and sends it to the master. The master then aggregates the messages to produce the full gradient $\nabla F(W^t) = \sum_{j=1}^{N} F_j(W^t)$, solves the linear subproblem by computing the leading right/left singular vectors of $\nabla F(W^t)$ and sends the solution back to the workers, who can form the next iterate W^{t+1}. NAIVE-DFW exactly mimics the behavior of the centralized FW algorithm, but induces a communication cost of $O(Ndm)$ per epoch as in many applications (such as those presented in Section 2.3) the local gradients are dense matrices. In the large-scale setting where the matrix dimensions d and m are both large, this cost dramatically limits the efficiency of the algorithm.

Singular Vector Averaging. A possible strategy to avoid this high communication cost is to ask each worker j to send to the master the rank-1 solution to the local version of the subproblem (3), in which they use their local gradient $\nabla F_j(W^t)$ as an estimate of the full gradient $\nabla F(W^t)$. This reduces the communication to a much more affordable cost of $O(N(d+m))$. Note that averaging the rank-1 solutions would typically lead to a rank-N update, which breaks the useful rank-1 property of FW and is undesirable when N is large. Instead, the master averages the singular vectors (weighted proportionally to n_j), resolving the sign ambiguity by setting the largest entry of each singular vector to be positive and using appropriate normalization, as mentioned for instance in Bro et al (2008). We refer to this strategy as Singular Vector Averaging (SVA). SVA is a reasonable heuristic when the individual functions are partitioned across nodes uniformly at random: in this case the local gradients can be seen as unbiased estimates of the full gradient. However the singular vector estimate itself is biased (averaging between workers only reduces its variance), and for n fixed this bias increases with the matrix dimensions d and m but also with the number of workers N (which is not a desirable property in the distributed setting). It is also expected to perform badly on arbitrary (non-uniform) partitions of functions across workers. Clearly, one cannot hope to establish strong convergence guarantees for SVA.

Algorithm 2 Our distributed algorithm DFW-Trace to solve (1)

Input: Initial point $W^0 \in \mathcal{D}$, number of iterations T
for $t = 0, \dots, T-1$ **do**
  Each worker j**:** $\nabla F_j(W^t) \leftarrow \sum_{i \in I_j} \nabla f_i(W^t)$
  All workers: draw the same $v_0 \in \mathbb{R}^m$ uniformly on unit sphere
  for $k = 0, \dots, K(t)-1$ **do** ▷ distributed power method
    Each worker j**:** send $u_{k+1,j} \leftarrow \nabla F_j(W^t) v_k$ to master
    Master: broadcast $u_{k+1} \leftarrow (\sum_{j=1}^N u_{k+1,j}) / \| \sum_{j=1}^N u_{k+1,j} \|$
    Each worker j**:** send $v_{k+1,j} \leftarrow \nabla F_j(W^t)^\top u_{k+1}$ to master
    Master: broadcast $v_{k+1} \leftarrow (\sum_{j=1}^N v_{k+1,j}) / \| \sum_{j=1}^N v_{k+1,j} \|$
  end for
  $\gamma^t \leftarrow \frac{2}{t+2}$ (or determined by line search) ▷ step size
  Each worker j**:** $W^{t+1} \leftarrow (1-\gamma^t) W^t - \gamma^t \mu u_{K(t)} v_{K(t)}^\top$ ▷ update
end for
Output: W^T

Table 1 Communication cost per epoch of the various algorithms. $K(t)$ is the number of power iterations used by DFW-Trace.

Algorithm	Communication cost	# communication rounds
Naive FW	Ndm	1
Singular Vector Averaging	$N(d+m)$	1
DFW-Trace	$2NK(t)(d+m)$	$2K(t)$

3.2 Proposed Approach

We now describe our proposed approach, referred to as DFW-Trace. We will see that DFW-Trace achieves roughly the small communication cost of SVA while enjoying a similar convergence rate as Naive-DFW (and hence centralized FW).

Algorithm. The main idea of DFW-Trace (Algorithm 2) is to solve the linear subproblem of FW approximately using a distributed version of the power method applied to the matrix $\nabla F(W^t)^\top F(W^t)$. At each outer iteration (epoch) t, the workers first generate a common random vector drawn uniformly on the unit sphere.[4] Then, for $K(t)$ iterations, the algorithm alternates between the workers computing matrix-vector products and the master aggregating the results. At the end of this procedure, workers hold the same approximate versions of the left and right singular vectors of $\nabla F(W^t)$ and use them to generate the next iterate W^{t+1}.

The communication cost of DFW-Trace per epoch is $O(NK(t)(d+m))$ (see Table 1 for a comparison with baselines). It is clear that as $K(t) \to \infty$, DFW-Trace computes the exact solution to the linear subproblems and hence has the same convergence guarantees as centralized FW. However, we would like to set $K(t) \ll \min(d, m)$ to provide a significant improvement over the $O(Ndm)$ cost of the naive distributed algorithm. The purpose of our analysis below is to show how to set $K(t)$ to preserve the convergence rate of the centralized algorithm.

[4] This can be done without communication: for instance, the workers can agree on a common random seed before running the algorithm.

Remark 2 (Other network topologies) Since any connected graph can be logically represented as a star graph by choosing a center, our method virtually works on any network (though it may incur additional communication). Depending on the topology, special care can be taken to reduce the communication overhead. An interesting case is the rooted tree network: we can adopt a hierarchical aggregation scheme which has the same communication cost of $O(NK(t)(d+m))$ as the star network but scales better to many workers by allowing parallel aggregations.[5] For a general graph with M edges, $O(MK(t)(d+m))$ communication is enough to broadcast the values to all workers so they can perform the aggregation locally.

Analysis. We will establish that for some appropriate choices of $K(t)$, DFW-Trace achieves sublinear convergence in expectation, as defined below.

Definition 1 Let $\delta \geq 0$ be an accuracy parameter. We say that DFW-Trace *converges sublinearly in expectation* if for each $t \geq 1$, its iterate W^t satisfies

$$\mathbb{E}[F(W^t)] - F(W^*) \leq \tfrac{2C_F}{t+2}(1+\delta), \tag{6}$$

where C_F is the curvature constant of F.

We have the following result.

Theorem 2 (Convergence) *Let F be a convex, differentiable function with curvature C_F and Lipschitz constant L w.r.t. the trace norm. For any accuracy parameter $\delta \geq 0$, the following properties hold for* DFW-Trace *(Algorithm 2):*

1. *If $m \geq 8$ and for any $t \geq 0$, $K(t) \geq 1 + \lceil \frac{\mu L(t+2)\ln m}{\delta C_F} \rceil$, then* DFW-Trace *converges sublinearly in expectation.*
2. *For any $t \geq 0$, let σ_1^t, σ_2^t be the largest and the second largest singular values of $\nabla F(W^t)$ and assume that σ_1^t has multiplicity 1 and there exists a constant β such that $\frac{\sigma_2^t}{\sigma_1^t} < \beta < 1$. If $K(t) \geq \max(\lceil \frac{\ln(\delta C_F) - \ln[m\mu L(t+2)]}{2\ln\beta} \rceil + 1, \widetilde{K})$ where $\widetilde{K}$ is a large enough constant,* DFW-Trace *converges sublinearly in expectation.*

Proof (Sketch) We briefly outline the main ingredients (see Appendix A for the detailed proof). We first show that if the linear subproblem is approximately solved in expectation (to sufficient accuracy), then the FW algorithm converges sublinearly in expectation. Relying on results on the convergence of the power method (Kuczyński and Woźniakowski, 1992) and on the Lipschitzness of F, we then derive the above results on the number of power iterations $K(t)$ needed to ensure sufficient accuracy under different assumptions. □

Theorem 2 characterizes the number of power iterations $K(t)$ at each epoch t which is sufficient to guarantee that DFW-Trace converges sublinearly in expectation to an optimal solution. Note that there are two regimes. The first part of the theorem establishes that if $K(t)$ scales linearly in t, the expected output of DFW-Trace after t epochs is a rank-t matrix with $O(1/t)$ approximation error (as in centralized FW, see Theorem 1). In the large-scale setting of interest, this implies that a good low-rank approximation can be achieved by running the algorithm for $t \ll \min(d, m)$ iterations, and with reasonable communication cost

[5] In Apache Spark, this is implemented in `treeReduce` and `treeAggregate`.

since $K(t) = O(t)$. Remarkably, this result holds without any assumption about the spectral structure of the gradient matrices. On the other hand, in the regime where the gradient matrices are "well-behaved" (in the sense that the ratio between their two largest singular values is bounded away from 1), the second part of the theorem shows that a much lower number of power iterations $K(t) = O(\log t)$ is sufficient to ensure the sublinear convergence in expectation. In Section 5, we will see experimentally on several datasets that this is indeed sufficient in practice to achieve convergence. We conclude this part with a few remarks mentioning some additional results, for which we omit the details due to the lack of space.

Remark 3 (Convergence in probability) We can also establish the sublinear convergence of DFW-Trace *in probability* (which is stronger than convergence in expectation). The results are analogous to Theorem 1 but require $K(t)$ to be quadratic in t for the first case, and linear in t for the second case.

Remark 4 (Constant number of power iterations) If we take the number of power iterations to be constant across epochs (i.e., $K(t) = K$ for all t), DFW-Trace converges in expectation to a neighborhood of the optimal solution whose size decreases with K. We can establish this by combining results on the approximation error of the power method with Theorem 5.1 in Freund and Grigas (2016).

3.3 Implementation

Our algorithm DFW-Trace (Algorithm 2) can be implemented as a sequence of map-reduce steps (Dean and Ghemawat, 2008). This allows the computation to be massively parallelized across the set of workers, while allowing a simple implementation and fast deployment on commodity and commercial clusters via existing distributed programming frameworks (Dean and Ghemawat, 2008; Zaharia et al, 2010). Nonetheless, some special care is needed if one wants to get an efficient implementation. In particular, it is key to leverage the fundamental property of FW algorithms that the updates are rank-1. This structural property implies that it is much more efficient to compute the gradient in a recursive manner, rather than from scratch using the current parameters. We use a notion of *sufficient information* to denote the local quantities (maintained by each worker) that are sufficient to compute the updates. This includes the local gradient (for the reason outlined above), and sometimes some quantities precomputed from the local dataset. Depending on the objective function and the relative size of the problem parameters n, m, d and N, the memory and/or time complexity may be improved by storing (some of) the sufficient information in low-rank form. We refer the reader to Appendix B for a concrete application of these ideas to the tasks of multi-task least square regression and multinomial logistic regression used in our experiments.

Based on the above principles, we developed an open-source Python implementation of DFW-Trace using the Apache Spark framework (Zaharia et al, 2010).[6] The package also implements the baseline strategies of Section 3.1, and currently uses dense representations. The code is modular and separates generic from task-specific components. In particular, the generic DFW-Trace algorithm is implemented in PySpark (Spark's Python API) in a task-agnostic fashion. On

[6] `https://github.com/WenjieZ/distributed-frank-wolfe`

the other hand, specific tasks (objective function, gradient, etc) are implemented separately in pure Python code. This allows users to easily extend the package by adding their own tasks of interest without requiring Spark knowledge. Specifically, the task interface should implement several methods: `stats` (to initialize the sufficient information), `update` (to update the sufficient information), and optionally `linesearch` (to use linesearch instead of default step size) and `loss` (to compute the value of the objective function). In the current version, we provide such interface for multi-task least square regression and multinomial logistic regression.

4 Related Work

There has been a recent surge of interest for the Frank-Wolfe algorithm and its variants in the machine learning community. The renewed popularity of this classic algorithm, introduced by Frank and Wolfe (1956), can be largely attributed to the work of Clarkson (2010) and more recently Jaggi (2013). They generalized its scope and showed that its strong convergence guarantees, efficient greedy updates and sparse iterates are valuable to tackle high-dimensional machine learning problems involving sparsity-inducing (non-smooth) regularization such as the L_1 norm and the trace norm. Subsequent work has extended the convergence results, for instance proving faster rates under some additional assumptions (see Lacoste-Julien and Jaggi, 2015; Garber and Hazan, 2015; Freund and Grigas, 2016).

As first-order methods, FW algorithms rely on gradients. In machine learning, computing the gradient of the objective typically requires a full pass over the dataset. To alleviate this computational cost on large datasets, some distributed versions of FW algorithms have recently been proposed for various problems. Bellet et al (2015) introduced a communication-efficient distributed FW algorithm for a class of problems under L_1 norm and simplex constraints, and provided an MPI-based implementation. Tran et al (2015) extend the algorithm to the Stale Synchronous Parallel (SSP) model. Moharrer and Ioannidis (2017) further generalized the class of problems which can be considered (still under L_1/simplex constraints) and proposed an efficient and modular implementation in Apache Spark (similar to what we propose in the present work for trace norm problems). Wang et al (2016) proposed a parallel and distributed version of the Block-Coordinate Frank-Wolfe algorithm (Lacoste-Julien et al, 2013) for problems with block-separable constraints. All these methods are designed for specific problem classes and do not apply to trace norm minimization. For general problems (including trace norm minimization), Wai et al (2017) recently introduced a decentralized FW algorithm in which workers communicate over a network graph without master node. The communication steps involve local averages of iterates and gradients between neighboring workers. In the master/slave distributed setting we consider, their algorithm essentially reduces to the naive distributed FW described in Section 3.1 and hence suffers from the high communication cost induced by transmitting gradients. In contrast to the above approaches, our work proposes a communication-efficient distributed FW algorithm for trace norm minimization.

Another direction to scale up FW algorithms to large datasets is to consider stochastic variants, where the gradient is replaced by an unbiased estimate computed on a mini-batch of data points (Hazan and Kale, 2012; Lan and Zhou, 2016; Hazan and Luo, 2016). The price to pay is a slower theoretical convergence rate,

and in practice some instability and convergence issues have been observed (see e.g., Liu and Tsang, 2017). The experimental results of Moharrer and Ioannidis (2017) show that current stochastic FW approaches do not match the performance of their distributed counterparts. Despite these limitations, this line of work is largely complementary to ours: when the number of workers N is small compared to the training set size n, each worker could compute an estimate of its local gradient to further reduce the computational cost. We leave this for future work.

We conclude this section by mentioning that other kinds of distributed algorithms have been proposed for special cases of our general problem (1). In particular, for the matrix completion problem, Mackey et al (2011) proposed a divide-and-conquer strategy, splitting the input matrix into submatrices, solving each subproblem in parallel with an existing matrix completion algorithm, and then combining the results.

5 Experiments

In this section, we validate the proposed approach through experiments on two tasks: multi-task least square regression and multinomial logistic regression (see Section 2.3). We use both synthetic and real-world datasets.

5.1 Experimental Setup

Environment. We run our Spark implementation described in Section 3.3 on a cluster with 5 identical machines, with Spark 1.6 deployed in standalone mode. One machine serves as the driver (master) and the other four as executors (workers). Each machine has 2 Intel Xeon E5645 2.40GHz CPUs, each with 6 physical cores. Each physical core has 2 threads. Therefore, we have 96 logical cores available as workers. The Spark cluster is configured to use all 96 logical cores unless otherwise stated. Each machine has 64GB RAM: our Spark deployment is configured to use 60GB, hence the executors use 240GB in total. The network card has a speed of 1Gb/s. The BLAS version does not enable multi-threading.

Datasets. For multi-task least square, we experiment on synthetic data generated as follows. The ground truth W has rank 10 and trace norm equal to 1 (we thus set $\mu = 1$ in the experiments). This is obtained by multiplying two arbitrary orthogonal matrices and a sparse diagonal matrix. X is generated randomly, with each coefficient following a Gaussian distribution, and we set $Y = XW$. We generate two versions of the dataset: a low-dimensional dataset ($n = 10^5$ samples, $d = 300$ features and $m = 300$ tasks) and a higher dimensional one ($n = 10^5$, $d = 1,000$ and $m = 1,000$). For multinomial logistic regression, we use a synthetic and a real dataset. The synthetic dataset has $n = 10^5$ samples, $p = 1,000$ features and $m = 1,000$ classes. The generation of W and X is the same as above, with the label vector y set to the one yielding the highest score for each point. The test set has 10^5 samples. Our real-world dataset is ImageNet from ILSVRC2012 challenge (Deng et al, 2009; Russakovsky et al, 2015), which has $n = 1,281,167$ training images in $m = 1,000$ classes. We use the learned features of dimension $p = 2048$ extracted from the deep neural network ResNet50 (He et al, 2016) provided by

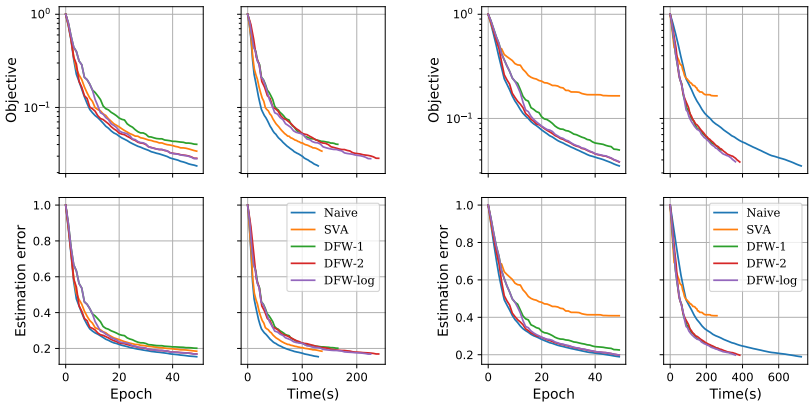

Fig. 1 Results for multi-task least square regression. Left: low-dimensional dataset ($n = 10^5$, $d = 300$ and $m = 300$). Right: higher-dimensional dataset ($n = 10^5$, $d = 1,000$ and $m = 1,000$).

Keras.[7] The validation set of the competition ($50,000$ images) serves as the test set.

Compared methods. We compare the following algorithms: Naive-DFW, SVA (the baselines described in Section 3.1) and three variants of our algorithm, DFW-Trace-1, DFW-Trace-2 and DFW-Trace-log (resp. using 1, 2 and $O(\log t)$ power iterations at step t). We have also experimented with DFW-Trace with $K(t) = O(t)$, but observed empirically that far fewer power iterations are sufficient in practice to ensure good convergence. We have also used SVA as warm start to the power iterations within DFW-Trace, which marginally improves the performance of DFW-Trace. We do not show these variants on the figures for clarity.

5.2 Results

Multi-task least square. For this task, we simply set the number of power iterations of DFW-Trace-log to $K(t) = \lfloor 1 + \log(t) \rfloor$. All algorithms use line search. Figure 1 shows the results for all methods on the low and high-dimensional versions of the dataset. The performance is shown with respect to the number of epochs and runtime, and for two metrics: the value of the objective function and the estimation error (relative Frobenius distance between the current W and the ground truth). On this dataset, the estimation error behaves similarly as the objective function. As expected, Naive-DFW performs the best with respect to the number of epochs as it computes the exact solution to the linear subproblem. On the low-dimensional dataset (left panel), it also provides the fastest decrease in objective/error. SVA also performs well on this dataset. However, when the dimension grows (right panel) the accuracy of SVA drops dramatically and Naive-DFW becomes much slower due to the increased communication cost. This confirms that these baselines do not scale well with the matrix dimensions. On the other hand, all variants of DFW-Trace perform much better than the baselines on the

[7] `https://github.com/fchollet/keras`

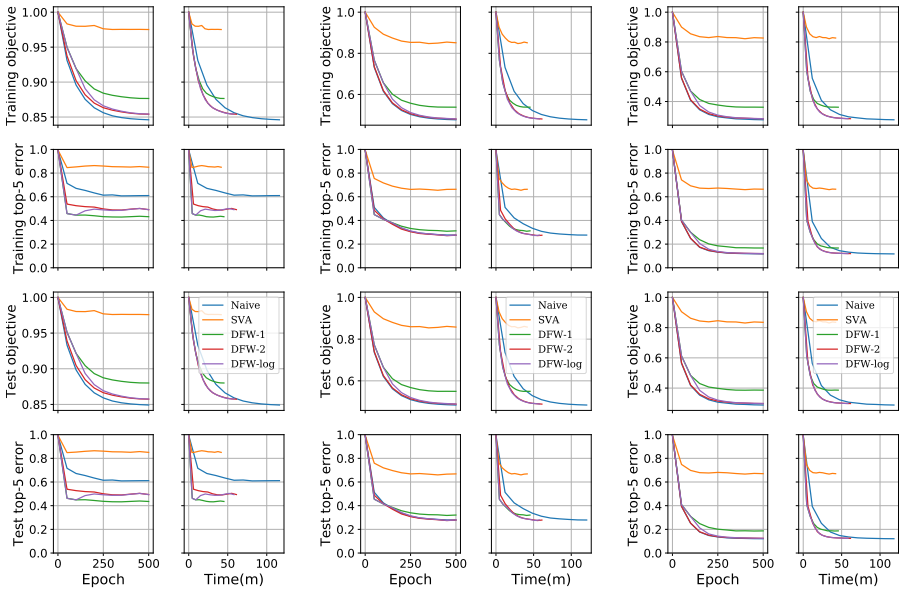

Fig. 2 Results for multinomial logistic regression (synthetic data) for several values of μ. Left: $\mu = 10$. Middle: $\mu = 50$. Right: $\mu = 100$. The error stands for the top-5 misclassification rate.

higher-dimensional dataset. This gap is expected to widen as the matrix dimensions increase. Remarkably, only 2 power iterations are sufficient to closely match the reduction in objective function achieved by the exact solution on this task. One can see the influence of the number of power iterations on the progress per epoch (notice for instance the clear break at iteration 10 when DFW-Trace-log switches from 1 to 2 power iterations), but this has a cost in terms of runtime. Overall, all variants of DFW-Trace reduce the objective/error at roughly the same speed. On a smaller scale version of the dataset, we verified that the gradients are well-behaved in the sense of Theorem 2: the average ratio between the two largest singular values over 100 epochs was found to be 0.86.

Multinomial logistic regression. Here, all algorithms use a fixed step size as there is no closed-form line search. As we observed empirically that this task requires a larger number of FW iterations to converge, we set $K(t) = \lfloor 1+0.5\log(t) \rfloor$ for DFW-Trace-log so that the number of power iterations does not exceed 2 as in the previous experiment. Figure 2 shows the results on the synthetic dataset for several values ofμ (the upper bound on the trace norm). They are consistent with those obtained for multi-task least square. In particular, SVA achieves converges to a suboptimal solution, while Naive-DFW converges fast in terms of epochs but its runtime is larger than DFW-Trace. DFW-Trace-2 and DFW-Trace-log perform well across all values of μ: this confirms that very few power iterations are sufficient to ensure good convergence. For more constrained problems ($\mu = 10$), the error does not align very well with the objective function and hence optimizing the subproblems to lower accuracy with DFW-Trace-1 works best.

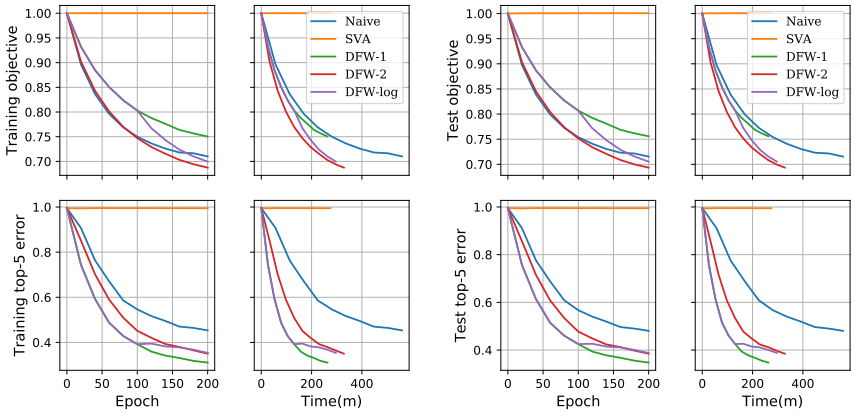

Fig. 3 Results for multinomial logistic regression (ImageNet dataset). The error stands for the top-5 misclassification rate.

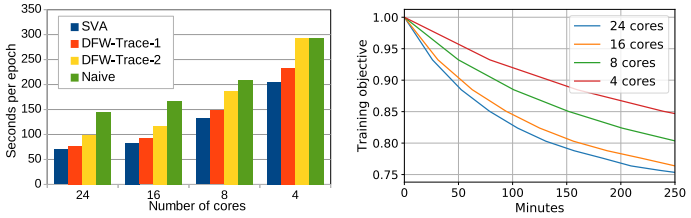

Fig. 4 Speed-ups with respect to the number of cores (ImageNet dataset). Left: time per epoch. Right: objective value with respect to runtime for DFW-Trace-1.

We now turn to the ImageNet dataset. The results for $\mu = 30$ with 24 cores are shown on Figure 3.[8] Again, the DFW-Trace variants clearly outperform Naive-DFW and SVA. While DFW-Trace-2 and DFW-Trace-log reduce the objective value faster than DFW-Trace-1, the latter reduces the error slightly faster. When run until convergence, all variants converge to state-of-the-art top-5 misclassification rate with these features (around 0.13, on par with the pre-trained deep neural net provided by Keras).

We conclude these experiments by investigating the speed-ups obtained when varying the number of cores on the ImageNet dataset. As seen on the left panel of Figure 4, the time per epoch nicely decreases with the number of cores (with diminishing returns, as expected in distributed computing). The right panel of Figure 4 illustrates this effect on the convergence speed for DFW-Trace-1.

6 Conclusion

In this work, we introduced a distributed Frank-Wolfe algorithm for learning high-dimensional low-rank matrices from large-scale datasets. Our DFW-Trace algo-

[8] The relative performance of the methods is the same for other values of μ. We omit these detailed results due to the lack of space.

rithm is communication-efficient, enjoys provable convergence rates and can be efficiently implemented in map-reduce operations. We implemented DFW-Trace as a Python toolbox relying on the Apache Spark distributed programming framework, and showed that it performs well on synthetic and real datasets.

In future work, we plan to investigate several directions. First, we would like to study whether faster theoretical convergence can be achieved under additional assumptions. Second, we wonder whether our algorithm can be deployed in GPUs and be used in neural networks with back-propagated gradients. Finally, we hope to explore how to best combine the ideas of distributed and stochastic Frank-Wolfe algorithms.

Acknowledgements This work was partially supported by ANR Pamela (grant ANR-16-CE23-0016-01) and by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. The first author would like to thank Ludovic Denoyer, Hubert Naacke, Mohamed-Amine Baazizi, and the engineers of LIP6 for their help during the deployment of the cluster.

References

Amit Y, Fink M, Srebro N, Ullman S (2007) Uncovering shared structures in multiclass classification. In: ICML

Argyriou A, Evgeniou T, Pontil M (2008) Convex multi-task feature learning. Machine Learning 73(3):243–272

Bach FR (2008) Consistency of trace norm minimization. Journal of Machine Learning Research 9:1019–1048

Bellet A, Liang Y, Garakani AB, Balcan MF, Sha F (2015) A Distributed Frank-Wolfe Algorithm for Communication-Efficient Sparse Learning. In: SDM

Bhojanapalli S, Neyshabur B, Srebro N (2016) Global Optimality of Local Search for Low Rank Matrix Recovery. In: NIPS

Bro R, Acar E, Kolda TG (2008) Resolving the sign ambiguity in the singular value decomposition. Journal of Chemometrics 22(2):135–140

Cabral R, De La Torre F, Costeira JP, Bernardino A (2013) Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decomposition. In: ICCV

Cabral RS, De la Torre F, Costeira JP, Bernardino A (2011) Matrix Completion for Multi-label Image Classification. In: NIPS

Cai JF, Candès EJ, Shen Z (2010) A singular value thresholding algorithm for matrix completion. SIAM Journal on Optimization 20(4):1956–1982

Candès EJ, Recht B (2009) Exact matrix completion via convex optimization. Foundations of Computational mathematics 9(6):717–772

Candès EJ, Tao T (2010) The power of convex relaxation: Near-optimal matrix completion. IEEE Transactions on Information Theory 56(5):2053–2080

Candes EJ, Eldar YC, Strohmer T, Voroninski V (2015) Phase retrieval via matrix completion. SIAM Review 57(2):225–251

Caruana R (1997) Multitask Learning. Machine Learning 28(1):41–75

Clarkson KL (2010) Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. ACM Transactions on Algorithms 6(4):63

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1):107–113

Deng J, Dong W, Socher R, Li LJ, Li K, Li FF (2009) ImageNet: A large-scale hierarchical image database. In: CVPR

Frank M, Wolfe P (1956) An algorithm for quadratic programming. Naval research logistics quarterly 3(1-2):95–110

Freund RM, Grigas P (2016) New analysis and results for the Frank–Wolfe method. Mathematical Programming 155(1–2):199–230

Garber D, Hazan E (2015) Faster Rates for the Frank-Wolfe Method over Strongly-Convex Sets. In: ICML

Goldberg A, Recht B, Xu J, Nowak R, Zhu X (2010) Transduction with matrix completion: Three birds with one stone. In: NIPS

Gross D (2011) Recovering low-rank matrices from few coefficients in any basis. IEEE Transactions on Information Theory 57(3):1548–1566

Gross D, Liu YK, Flammia ST, Becker S, Eisert J (2010) Quantum state tomography via compressed sensing. Physical review letters 105(15):150,401

Harchaoui Z, Douze M, Paulin M, Dudik M, Malick J (2012) Large-scale image classification with trace-norm regularization. In: CVPR

Harchaoui Z, Juditsky A, Nemirovski A (2015) Conditional gradient algorithms for norm-regularized smooth convex optimization. Mathematical Programming 152(1–2):75–112

Hazan E (2008) Sparse approximate solutions to semidefinite programs. In: Latin American Symposium on Theoretical Informatics

Hazan E, Kale S (2012) Projection-free Online Learning. In: ICML

Hazan E, Luo H (2016) Variance-Reduced and Projection-Free Stochastic Optimization. In: ICML

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: CVPR

Jaggi M (2013) Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In: ICML

Jaggi M, Sulovsk M, et al (2010) A simple algorithm for nuclear norm regularized problems. In: ICML

Ji H, Liu C, Shen Z, Xu Y (2010) Robust video denoising using low rank matrix completion. In: CVPR

Koltchinskii V, Lounici K, Tsybakov AB (2011) Nuclear-norm penalization and optimal rates for noisy low-rank matrix completion. The Annals of Statistics 39(5):2302–2329

Koren Y, Bell R, Volinsky C, et al (2009) Matrix factorization techniques for recommender systems. Computer 42(8):30–37

Kuczyński J, Woźniakowski H (1992) Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. SIAM Journal on Matrix Analysis and Applications 13(4):1094–1122

Lacoste-Julien S, Jaggi M (2015) On the Global Linear Convergence of Frank-Wolfe Optimization Variants. In: NIPS

Lacoste-Julien S, Jaggi M, Schmidt M, Pletscher P (2013) Block-Coordinate Frank-Wolfe Optimization for Structural SVMs. In: ICML

Lan G, Zhou Y (2016) Conditional Gradient Sliding for Convex Optimization. SIAM Journal on Optimization 26(2):1379–1409

Liu Z, Tsang I (2017) Approximate Conditional Gradient Descent on Multi-Class Classification. In: AAAI

Ma S, Goldfarb D, Chen L (2011) Fixed point and Bregman iterative methods for matrix rank minimization. Mathematical Programming 128(1-2):321–353
Mackey LW, Jordan MI, Talwalkar A (2011) Divide-and-conquer matrix factorization. In: Advances in Neural Information Processing Systems, pp 1134–1142
Moharrer A, Ioannidis S (2017) Distributing Frank-Wolfe via Map-Reduce. In: ICDM
Parikh N, Boyd S (2013) Proximal algorithms. Foundations and Trends in Optimization 1(3):123–231
Pong TK, Tseng P, Ji S, Ye J (2010) Trace norm regularization: Reformulations, algorithms, and multi-task learning. SIAM Journal on Optimization 20(6):3465–3489
Recht B (2011) A simpler approach to matrix completion. Journal of Machine Learning Research 12:3413–3430
Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision 115(3):211–252
Sturm JF (1999) Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optimization methods and software 11(1-4):625–653
Toh KC, Todd MJ, Tütüncü RH (1999) SDPT3—a MATLAB software package for semidefinite programming, version 1.3. Optimization methods and software 11(1-4):545–581
Tran NL, Peel T, Skhiri S (2015) Distributed frank-wolfe under pipelined stale synchronous parallelism. In: IEEE Big Data
Wai HT, Lafond J, Scaglione A, Moulines E (2017) Decentralized Frank-Wolfe Algorithm for Convex and Non-convex Problems. IEEE Transactions on Automatic Control 62:5522–5537
Wang YX, Sadhanala V, Dai W, Neiswanger W, Sra S, Xing E (2016) Parallel and distributed block-coordinate Frank-Wolfe algorithms. In: ICML
Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: Cluster Computing with Working Sets. In: HotCloud

Appendix A Proof of Theorem 2

Notice that our distributed version of the power method used in DFW-Trace (Algorithm 2, lines 5–10) exactly corresponds to the serial power method applied to the full gradient $\nabla F(W^t)$. Hence DFW-Trace performs the same steps as a centralized Frank-Wolfe algorithm that would use the power method to approximately solve the subproblems. We will thus abstract away the details related to the distributed setting (e.g., how the data is split, how parallel computation is organized): our analysis consists in characterizing the approximation error incurred by the power method and showing that this error is small enough to ensure that the Frank-Wolfe algorithm converges in expectation.

We start by establishing that if the linear subproblem is approximately solved in expectation (to sufficient accuracy), then the standard Frank-Wolfe algorithm converges sublinearly in expectation (in the sense of Definition 1).

Lemma 1 *Let $\delta \geq 0$ be an accuracy parameter. If at each step $t \geq 0$, the linear subproblem is approximately solved in expectation, i.e. we find a random variable $\hat{S}$ such that*

$$\langle \mathbb{E}[\hat{S}|W^t], \nabla F(W^t)\rangle \leq \min_{S\in\mathcal{D}} \langle S, \nabla F(W^t)\rangle + \tfrac{1}{2}\delta\gamma^t C_F, \tag{7}$$

then the Frank-Wolfe algorithm converges sublinearly in expectation.

Proof At any step t, given W^t we set $W^{t+1} = W^t + \gamma^t(\hat{S} - W^t)$ with arbitrary step size $\gamma^t \in [0, 1]$. From the definition of the curvature constant C_F (Jaggi, 2013):

$$F(W^{t+1}) \leq F(W^t) + \gamma^t\langle \hat{S} - W^t, \nabla F(W^t)\rangle + \tfrac{(\gamma^t)^2}{2} C_F.$$

We can now take conditional expectation on both sides and use (7) to get

$$\begin{aligned}
\mathbb{E}[F(W^{t+1})|W^t] &\leq F(W^t) + \gamma^t\langle \mathbb{E}[\hat{S}|W^t] - W^t, \nabla F(W^t)\rangle + \tfrac{(\gamma^t)^2}{2} C_F \\
&\leq F(W^t) + \gamma^t \left(\min_{S\in\mathcal{D}} \langle S - W^t, \nabla F(W^t)\rangle\right) + \tfrac{(\gamma^t)^2}{2} C_F(1+\delta) \\
&\leq F(W^t) - \gamma^t G(W^t) + (\gamma^t)^2 C,
\end{aligned}$$

where we denote $G(W) := \max_{S\in\mathcal{D}} \langle W - S, \nabla F(W)\rangle$ and $C := \frac{C_F}{2}(1+\delta)$. The function $G(W)$ is known as the *duality gap* and satisfies $F(W) - F(W^*) \leq G(W)$ — see Jaggi (2013) for details. Denoting $H(W) := F(W) - F(W^*)$, we have

$$\begin{aligned}
\mathbb{E}[H(W^{t+1})|W^t] &\leq H(W^t) - \gamma^t G(W^t) + (\gamma^t)^2 C \\
&\leq H(W^t) - \gamma^t H(W^t) + (\gamma^t)^2 C \\
&= (1 - \gamma^t) H(W^t) + (\gamma^t)^2 C,
\end{aligned}$$

where we use the duality $H(x) \leq G(x)$.

We shall use induction over t to prove the sublinear convergence in expectation (6), i.e., we want to show that

$$\mathbb{E}[H(W^t)] \leq \tfrac{4C}{t+2}, \quad \text{for } t = 1, 2, ...$$

We prove this for the default step size $\gamma^t = \frac{2}{t+2}$ (we can easily prove the same thing for the line search variant, as the resulting iterates always achieve a lower objective than with the default step size). For $t = 1$, we have $\gamma^0 = \frac{2}{0+2} = 1$. For any $W \in \mathcal{D}$, we have $H(W) \leq \frac{C_F}{2} < C < \frac{4}{3}C$. This proves the case of $t = 1$. Consider now $t \geq 2$, then

$$\begin{aligned}
\mathbb{E}[H(W^{t+1})] = \mathbb{E}[\mathbb{E}[H(W^{t+1})|W^t]] &\leq (1 - \gamma^t)\mathbb{E}[H(W^t)] + (\gamma^t)^2 C \\
&\leq \left(1 - \tfrac{2}{t+2}\right)\tfrac{4C}{t+2} + \left(\tfrac{2}{t+2}\right)^2 C.
\end{aligned}$$

Simply rearranging the terms gives

$$\mathbb{E}[H(W^{t+1})] \leq \tfrac{4(t+1)C}{(t+2)^2} < \tfrac{4(t+1)C}{(t+1)(t+3)} = \tfrac{4C}{t+3}.$$

This concludes the proof. □

Based on Lemma 1, in order to prove Theorem 2 we need to quantify the number of power method iterations needed to achieve the desired accuracy (7) for the linear subproblems. We will rely on some results from Kuczyński and Woźniakowski (1992, Theorem 3.1 therein), which we recall in the lemma below.

Lemma 2 (Kuczyński and Woźniakowski, 1992) *Let $A \in \mathbb{R}^{m\times m}$ be any symmetric and positive definite matrix, and b be a random vector chosen uniformly on the unit sphere (with P the corresponding probability measure). Denote by λ_1 the largest eigenvalue of A and by $\xi = \xi(A, b, K)$ the estimate given by K power iterations. We define its average relative error $e(\xi)$ as*

$$e(\xi) := \int_{\|b\|=1} \left| \frac{\xi - \lambda_1}{\lambda_1} \right| P(db).$$

Then for any $K \geq 2$ and $m \geq 8$, regardless of A, we have

$$e(\xi) \leq \alpha(m) \frac{\ln m}{K-1},$$

where $\pi^{-1/2} \leq \alpha(m) \leq 0.871$ and, for large m, $\alpha(m) \approx \pi^{-1/2} \approx 0.564$.

Moreover, if λ has multiplicity 1, denoting the second largest eigenvalue by λ_2, then there exists a constant $\widetilde{K}$, so that for any $K > \widetilde{K}$, we have

$$e(\xi) \leq m \left(\frac{\lambda_2}{\lambda_1} \right)^{K-1}.$$

We introduce a last technical lemma.

Lemma 3 *If a differentiable function F is L-Lipschitz continuous w.r.t. the trace norm, then for any matrix W, all singular values of $\nabla F(W)$ are smaller than L.*

Proof For any matrix W, the definition of L-Lipschitzness implies that

$$\sup_{\Delta W \neq 0} \frac{|F(W + \Delta W) - F(W)|}{\|\Delta W\|_*} \leq L.$$

According to the mean value theorem, there exists a matrix X between W and $W + \Delta W$ such that

$$\sup_{\Delta W \neq 0} \left\langle \nabla F(X), \frac{\Delta W}{\|\Delta W\|_*} \right\rangle \leq L.$$

Denote the largest singular value of W by $\sigma_1(W)$. Since the spectral norm is the dual norm of the trace norm, we have $\sigma_1(\nabla F(X)) \leq L$. Letting $\Delta W \to 0$, we get $\sigma_1(\nabla F(W)) \leq L$. □

Based on the above intermediary results, we can now prove Theorem 2. For any $t \geq 0$, denote $A^t := \nabla F(W^t)$. The largest eigenvalue of ${A^t}^\top A^t$ is the square of the largest singular value of A^t, denoted as σ_1^t. We estimate $(\sigma_1^t)^2$ as $v_{K(t)}^\top {A^t}^\top A^t v_{K(t)}$, where $v_{K(t)}$ is the normalized unit vector after $K(t)$ power iterations. We also denote $u_{K(t)} := A^t v_{K(t)} / \|A^t v_{K(t)}\|$.

According to Lemma 2, we have

$$\mathbb{E} \left| \frac{v_{K(t)}^\top {A^t}^\top A^t v_{K(t)} - (\sigma_1^t)^2}{(\sigma_1^t)^2} \right| \leq \frac{\ln m}{K(t) - 1}.$$

Therefore:

$$\mathbb{E}\left|\frac{\|A^t v_{K(t)}\|}{\sigma_1^t}-1\right| \leq \mathbb{E}\left|\frac{\|A^t v_{K(t)}\|}{\sigma_1^t}-1\right|\left|\frac{\|A^t v_{K(t)}\|}{\sigma_1^t}+1\right| = \mathbb{E}\left|\frac{\|A^t v_{K(t)}\|^2}{(\sigma_1^t)^2}-1\right| \leq \frac{\ln m}{K(t)-1}.$$

Let $K(t) = 1 + \lceil \frac{\mu L(t+2)\ln m}{\delta C_F} \rceil$, we get

$$\mathbb{E}\left|\frac{\|A^t v_{K(t)}\|}{\sigma_1^t}-1\right| \leq \frac{1}{2}\frac{\delta\gamma^t C_F}{\mu L} \leq \frac{1}{2}\frac{\delta\gamma^t C_F}{\mu\sigma_1^t},$$

where the last inequality uses Lemma 3.

Removing the absolute sign and the denominator, we get

$$\mathbb{E}[\mu(\sigma_1^t - \|A^t v_{K(t)}\|)] \leq \tfrac{1}{2}\delta\gamma^t C_F.$$

Rearranging the terms, we obtain

$$-\mu\mathbb{E}\|A^t v_{K(t)}\| \leq -\mu\sigma_1^t + \tfrac{1}{2}\delta\gamma^t C_F. \tag{8}$$

On the other hand, we have

$$\mathbb{E}\|A^t v_{K(t)}\| = \mathbb{E}\left[\frac{v_{K(t)}^\top {A^t}^\top A^t v_{K(t)}}{\|A^t v_{K(t)}\|}\right] = \mathbb{E}[u_{K(t)}^\top A^t v_{K(t)}] = \mathbb{E}\left\langle u_{K(t)} v_{K(t)}^\top, A^t\right\rangle, \tag{9}$$

and

$$\mu\sigma_1^t = \max_{\|S\|_* \leq \mu} \left\langle S, A^t\right\rangle. \tag{10}$$

Replacing (9) and (10) into (8), we obtain (7). The first assertion of Theorem 2 thus holds by application of Lemma 1. For the second assertion, the proof is nearly identical. Indeed, by replacing $\frac{\ln m}{K(t)-1}$ with $m\beta^{2K(t)-2}$, we get the desired result.

Appendix B Implementation Details for Two Tasks

For the two tasks studied in this paper, we describe the *sufficient information* maintained by workers and how to efficiently update it. Table 2 summarizes the per-worker time and memory complexity of DFW-Trace depending on the representation used for the sufficient information. Generally, the low-rank representation is more efficient when the number of local data points $n_j < \min(d, m)$.

Multi-task least square regression. Recalling the multi-task regression formulation in (4), for any worker j we will denote by X_j the $n_j \times d$ matrix representing the feature representation of the data points held by j. Similarly, we use Y_j to denote the $n_j \times m$ response matrix associated with these data points. The gradient of (4) is given by $\nabla F(W) = X^\top(XW - Y)$. At each step t, each worker j will store $(X_j^\top Y_j, X_j^\top X_j, X_j^\top X_j W^t, W^t, \nabla F_j(W^t))$ as sufficient information. The quantities $X_j^\top Y_j$ and $X_j^\top X_j$ are fixed and precomputed. Given W^t, $W^{t+1} = (1-\gamma^t)W^t + \gamma^t S^t$

Table 2 Time and memory complexity of DFW-Trace for the j-th worker on two tasks with dense vs. low-rank representations for the sufficient information.

	Multi-task least square		**Multinomial logistic regression**	
	Dense	Low-rank	Dense	Low-rank
Init.	$O(n_j(d^2+md))$	0	$O(n_jd+md)$	0
Power iter.	$O(md)$	$O(n_j(d+m))$	$O(md)$	$O(n_j(d+m))$
Update	$O(d^2+md)$	$O(n_j(d+m))$	$O(n_jmd)$	$O(n_j(d+m))$
Line search	$O(d^2+md)$	$O(n_jm)$	—	—
Memory	$O(d^2+md)$	$O(n_j(d+m))$	$O(n_j(d+m)+md)$	$O(n_j(d+m))$

is efficiently obtained by rescaling W^t and adding the rank-1 matrix $\gamma^t S^t$. A similar update scheme is used for $X_j^\top X_j W^t$. Assuming W^0 is initialized to the zero matrix, the local gradient is initialized as $\nabla F_j(W^0) = -X_j^\top Y_j$ and can be efficiently updated using the following formula:

$$\begin{aligned}\nabla F_j(W^{t+1}) &= X_j^\top (X_j W^{t+1} - Y_j) = X_j^\top (X_j[(1-\gamma^t)W^t + \gamma^t S^t] - Y_j) \\ &= (1-\gamma^t)\nabla F_j(W^t) + \gamma^t (X_j^\top X_j S^t - X_j^\top Y_j).\end{aligned}$$

The same idea can be applied to perform line search, as the optimal step size at any step t is given by the following closed-form formula:

$$\gamma^t = \frac{\langle -\nabla f(W^t), S^t - W^t \rangle}{\langle X^\top X(S^t - W), S^t - W \rangle}.$$

Multinomial logistic regression. We now turn to the multi-class classification problem (5). As above, for a worker j we denote by X_j its local $n_j \times d$ feature matrix and by $Y_j \in \mathbb{R}^{n_j}$ the associated labels. The gradient of (5) is given by $\nabla F(W) = X^\top (P - H)$, where P and H are $n \times m$ matrices whose entries (i, l) are $P_{il} = \frac{\exp(w_l^T x_i)}{\sum_k \exp(w_k^T x_i)}$ and $H_{il} = \mathbb{I}[y_i = l]$ respectively. The sufficient information stored by worker j at each step t is $(X_j, X_j^\top H_j, X_j W^t, \nabla F_j(W^t))$. $X_j^\top H_j$ is fixed and precomputed. Assuming that W^0 is the zero matrix, $X_j W^t$ is initialized to zero and easily updated through a low-rank update. The local gradient $\nabla F_j(W^t) = X_j^\top P_j - X_j^\top H_j$ can then be obtained by applying the softmax operator on $X_j W^t$. Note that there is no closed-form for the line search.